

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re the Application of:  
Eitan Marcus et. al

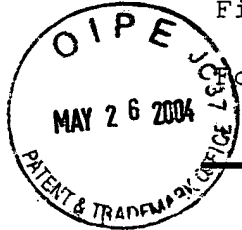
Application No.: 10/040940

Filed: 01/09/02

Art Unit: 2863

For: Adaptive Test Generation

Examiner: Xiuqin Sun



Mail Stop Non-Fee Amendment  
5 Commissioner for Patents  
PO Box 1450  
Alexandria, VA 22313-1450

DECLARATION UNDER 37 CFR 1.131

10 Sir:

We, the undersigned, Allon Adir, Roy Emek, and Eitan Marcus hereby declare as follows:

15 1) We are the Applicants in the patent application identified above, and are co-inventors of the subject matter described and claimed in claims 12-14, 16-19, 29-34, 36-39, 41-47, and 49-52 therein.

20 2) Prior to January 3, 2002, we conceived our invention, as described and claimed in the subject Application, in Israel, a WTO member country. Conception of the invention is evidenced by a marked-up draft copy of the disclosure of the Application, which was included in an electronic mail communication transmitted by Eitan Marcus prior to January 3, 2002, and which is attached hereto as Appendix A. Appendix A at page A-1 includes a  
25 copy of the electronic mail communication, demonstrating an at-

IL9-2001-0019

42045

## Declaration under 37 C.F.R. 131

tached Zip archive named "42045.zip". Appendix A at page A-2 includes a listing of the content of the Zip archive 42045.zip, indicating an archived Microsoft Word file named "42045S06.doc". Appendix A at page A-3 et seq. is a printout of the file 42045S06.doc, which is the draft copy of the Application hereof.

3) Dates deleted from Appendix A are prior to January 3, 2002.

4) The electronic mail communication of Appendix A was transmitted to Arthur S. Bickel, of Sanford T. Colb & Co., who was retained by International Business Machines Corporation (IBM) as outside counsel for the purpose of preparing the present patent application.

5) The following table shows the correspondence between the elements of claims 12-14, 16-19, 29-34, 36-39, 41-47, and 49-52 in the present patent application and statements in the Disclosure shown in Appendix A:

<b>Claims (Application)</b>	<b>Disclosure (Appendix A)</b>
Claim 12	Claim 16
Claim 13	Claim 17; para. [0039]
Claim 14	Claim 18
Claim 16	Claim 21
Claim 17	Claim 22
Claim 18	Claim 23
Claim 19	Claim 24
Claim 29	Claim 34
Claim 30	Claim 35; para. [0039]
Claim 31	Claim 36
Claim 32	Claim 37
Claim 33	Claim 38

42045

## Declaration under 37 C.F.R. 131

Claim 34	Claim 39
Claim 36	Claim 41
Claim 37	Claim 42
Claim 38	Claim 43
Claim 39	Claim 44
Claim 41	Claim 46
Claim 42	Claim 47
Claim 43	Claim 48
Claim 44	Claim 49
Claim 45	Claim 50
Claim 46	Claim 51
Claim 47	Claim 52
Claim 49	Claim 54
Claim 50	Claim 55
Claim 51	Claim 56
Claim 52	Claim 57

This table demonstrates that we conceived the entire invention, as recited in claims 12-14, 16-19, 29-34, 36-39, 41-47, and 49-52, prior to January 3, 2002.

5

6) On January 2, 2002, Dr. Bickel sent us a revised draft of the Application. On about January 3, 2002 we approved the revised draft of the Application, and it was then sent to the United States, where it was filed on January 9, 2002.

10

7) At the time the invention was made, we were obligated to assign the claimed invention to IBM.

8) We hereby declare that all statements made herein of our own knowledge are true and that all statements made on information and conjecture are thought to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the

IL9-2001-0019

42045

## Declaration under 37 C.F.R. 131

United States Code and that such willful false statements may jeopardize the validity of the application of any patent issued thereon.

אלון אדיר  
Allon Adir, Citizen of Israel,  
70 Allonim Street, Tivon, Is-  
rael

Date:

2-5-2004

איתן מרקוס  
Eitan Marcus, Citizen of Is-  
rael, ~~5 Eder Street~~, Haifa, Is-  
rael 9 Greenberg St

Date:

2-5-2004

רוי עמק  
Roy Emek, Citizen of Israel,  
~~32 Oranim Street, Kfar~~  
~~Shmaryahu, Israel~~

Date:

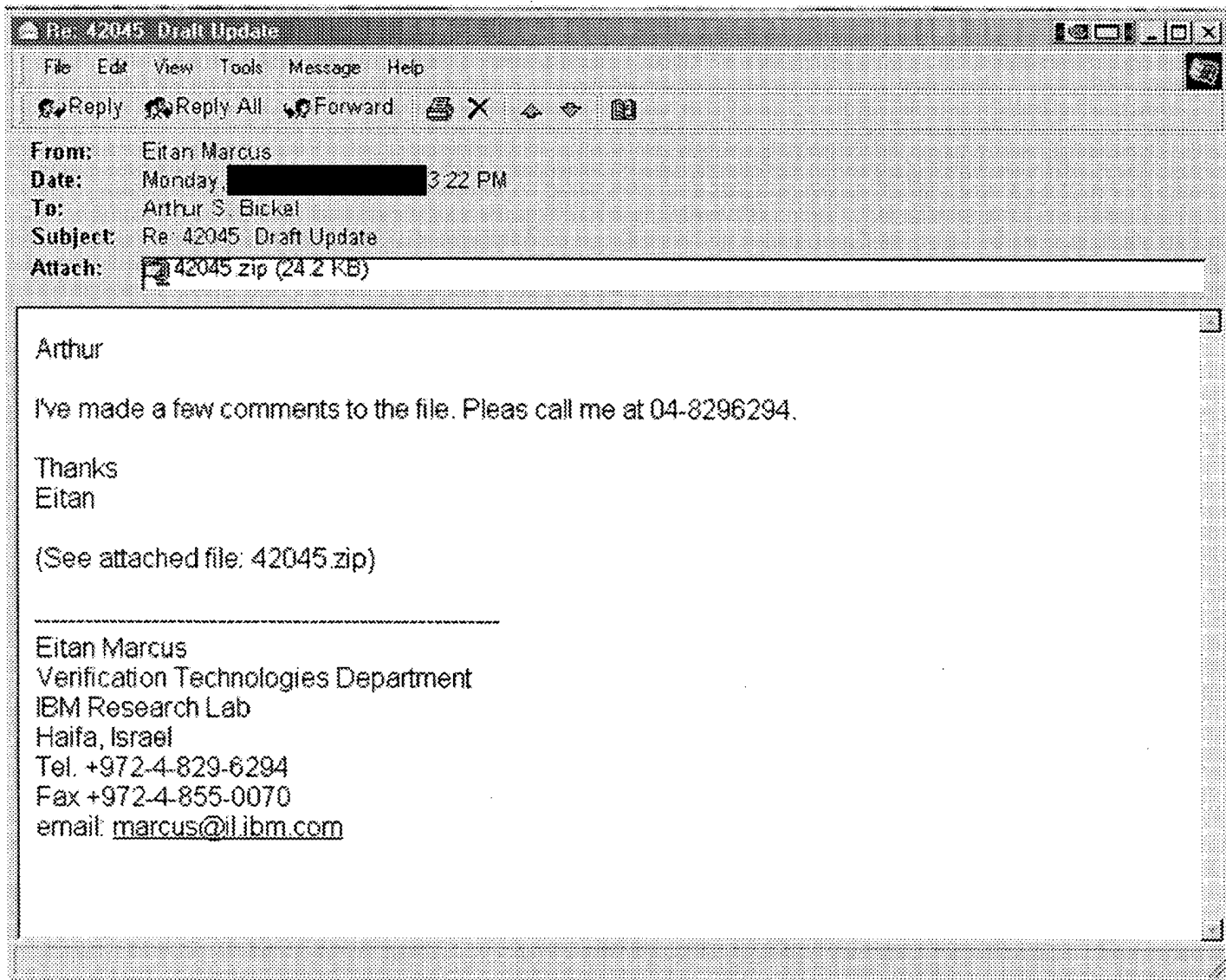
2-May-04

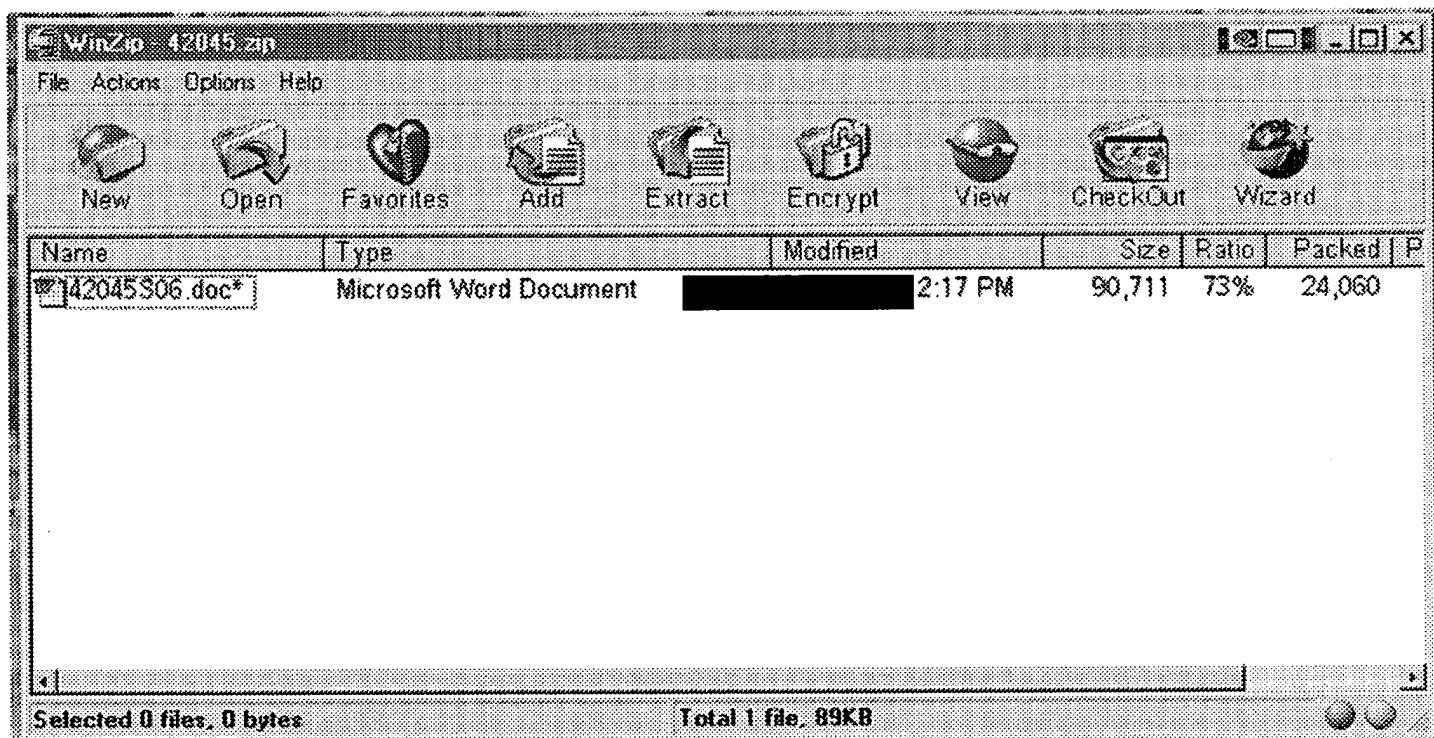
new address:  
10 Matmon St,  
Tel Aviv, Israel.

42045

Declaration under 37 C.F.R. 131

APPENDIX A





## Adaptive Test Program generation

### BACKGROUND OF THE INVENTION

#### 1. Field of the Invention.

5 [0001.] This invention generally relates to test program generation, and more specifically relates to a computer mechanism and method for testing a design for compliance with the design specification.

#### 2. Description of the Related Art.

10 [0002.] The extension of modern information technology into everyday life is due in large part to the existence, functionality and relatively low cost of advanced integrated circuits, and in particular to advancements in computer processors. As technology has advanced, the sophistication of both the hardware and software of computer systems have greatly increased. An important aspect of designing an advanced computer system is the ability to thoroughly test its design to assure that the design complies with desired specifications. [EM: This may be pedantic, but there are alternative ways to do verification that don't require tests] using this technique, Such verification requires the generation of a large number of instruction sequences to assure that the system behaves properly under a wide variety of circumstances.

20 [0003.] Test program generators are basically sophisticated software engines, which are used to create numerous test cases. By appropriate configuration, it is possible for test program generation to be focused on very specific ranges of conditions, or broadened to cover a wide range of logic. Today, large numbers of test cases can be created in the time that a single test case could be written manually, as was done prior to the advent of test  
25 program generators.

5       **[0004.]** Typically, the input to the test program generator is a sequence of partially specified instructions, which acts as a template for the test to be generated. The instruction stream is generally incomplete, in that various details of each instruction, such as the specific source and the target re-  
10       sources to be used, the data values of each uninitialized resource, and even the type of the instruction to be generated, may be left unspecified. The test program generator then generates a complete test by filling in the missing information with valid combinations of random values, or values that it as-  
15       sumes to be likely to generate an interesting test. The choice of random values is often biased, either explicitly in the input file, or through testing knowl-  
20       edge known to the test program generator, so as to increase the likelihood of detecting a design flaw. The use of templates in this manner allows the crea-  
25       tion of numerous test cases that stress the implementation of the logic of the design.

15       **[0005.]** ~~Early test program generators were static, in that they worked without any knowledge of the running state of the verified design during the generation of the test. Without this knowledge, it was necessary for the test program generator to guess the state when considering its choice of legal or interesting values.~~

20       **[0006.]** ~~Dynamic test program generators were subsequently developed, which simulate run a simulation of the test as it is being generated.~~ The in-  
25       formation obtained from the simulation is used to guide the test program generator, without forcing it to resort to extensive speculation as to what values may be desirable.

25       **[0007.]** During the past decade, model-based random test program generators have become popular in processor architectural design verification

and software testing. An example of such a random test generators include the IBM tool, "Genesys", which is disclosed in the document *Model-Based Test Generation for Process Design Verification*, Y. Lichtenstein *et al.*, Sixth Innovative Applications of Artificial Intelligence Conference, August 1994, pp. 83-94.

**[0008.]** Another conventional test generator, AVPGEN, is disclosed in the document *AVPGEN - A Generator for Architecture Verification Test Cases*, A. Chandra, *et al.* IEEE Trans. Very Large Scale Integration (VLSI) Syst. 3, No. 2, 188-200 (June 1995).

**[0009.]** Known test program generators are limited in their abilities to adapt to unexpected situations encountered during the test, and offer at best ad hoc, solutions to handle situations such as program interrupts.

#### SUMMARY OF THE INVENTION

**[0010.]** It is an advantage of some aspects of the present invention that an event handling mechanism be events are provided in a test program generator. The events constitute a single, uncomplicated mechanism for dealing with disparate exceptional situations that may be encountered during test program generation.

**[0011.]** It is another advantage of some aspects of the present invention that beneficial situations encountered during test program generation can be recognized and exploited.

**[0012.]** It is yet another advantage of some aspects of the present invention that detrimental situations or failures encountered during test program generation can be recognized and avoided before they happen.

**[0013.]** It is a further advantage of some aspects of the present invention that events to be recognized during test program generation can be defined without modification or recompilation of a test program generator.

5 **[0014.]** It is yet a further advantage of some aspects of the present invention that the body of events defines an alternative input stream to be generated, and uses ~~is defined using~~ the same language as the one used for generating the main instruction stream.

10 **[0015.]** These and other advantages of the present invention are attained by a test program generator that produces test instructions according to specification of a design being verified. The instructions are typically generated randomly, at least in part. The system is capable of interpreting defined events, detecting an impending occurrence of an event by evaluating the condition of the event, and responding to the event by generating an alternate instruction stream defined as part of the event.

15 **[0016.]** An important consequence of using randomly generated test cases is that the execution of the test may lead to situations that cannot be anticipated at the time the input instruction stream is written. Examples of such events include (1) program interrupts, (2) the event that there is insufficient continuous unoccupied memory to store subsequent instructions, and  
20 (3) false branch prediction.

**[0017.]** Some of the above noted events, for example the event that there is insufficient continuous unoccupied memory are detrimental, because they are likely to cause failure of test program generation. Other events may be viewed as beneficial, in that the situation that arises on their occurrence  
25 can be exploited to increase the likelihood of finding design flaws.

5 [0018.] In order to deal with events such as the exemplary events noted above, it is desirable to generate an alternative instruction stream. In the case of detrimental events, the approach is to detect the situation prior to its actual occurrence, and issuing appropriate instructions to avoid its occurrence. Beneficial events also need to be detected, so that alternative instructions may be inserted into the test to exploit the situation.

10 [0019.] The ability of users to detect the occurrence of triggering conditions of events and to inject an alternative instruction stream into the test program is referred to herein as "adaptive test program generation". Using adaptive test program generation, users can define multiple input streams for the test program generator that are reactive to the running states of the generation. Because of randomness, such states cannot be known at the time the input is ??? defind.

15 {Textual version of claims to be inserted here once they are agreed upon}

#### BRIEF DESCRIPTION OF THE DRAWINGS

20 [0020.] For a better understanding of these and other objects of the present invention, reference is made to the detailed description of the invention, by way of example, which is to be read in conjunction with the following drawings, wherein:

[0021.] Fig. 1 is a block diagram of a test program generator that is operable in accordance with a preferred embodiment of the invention;

25 [0022.] Fig. 2 is a block diagram of an input file structure in accordance with a preferred embodiment of the invention;

**[0023.]** Fig. 3 is a block diagram illustrating the structure of an event, which is operative in the test program generator shown in Fig. 1 in accordance with a preferred embodiment of the invention;

**[0024.]** Fig. 4 is a block diagram illustrating a test generator engine of the test program generator shown in Fig. 1 in accordance with a preferred embodiment of the invention; and

**[0025.]** Fig. 5 is a flow chart of a method of test program generation, which is operative in accordance with a preferred embodiment of the invention.

#### 10 **DESCRIPTION OF THE PREFERRED EMBODIMENT**

**[0026.]** In the following description, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent however, to one skilled in the art that the present invention may be practiced without these specific details. In other instances well known circuits, control logic, and the details of computer program instructions for conventional algorithms and processes have not been shown in detail in order not to unnecessarily obscure the present invention.

**[0027.]** Software programming code, which embodies aspects of the present invention, is typically maintained in permanent storage, such as a computer readable medium. In a client/server environment, such software programming code may be stored on a client or a server. The software programming code may be embodied on any of a variety of known media for use with a data processing system, such as a diskette, or hard drive, or CD-ROM. The code may be distributed on such media, or may be distributed to users from the memory or storage of one computer system over a network of some type to other computer systems for use by users of such other systems. The

techniques and methods for embodying software program code on physical media and distributing software code via networks are well known and will not be further discussed herein.

### **Definitions.**

- 5   **[0028.]** A "statement" is a single item ~~that defines a stream entity~~ for controlling generation of the test. Statements can either be "instruction statements" (partially specified instructions) which describe instructions to be generated, or "control statements" which direct the flow of control of other statements.
- 10   **[0029.]** {EM: This should be after paragraph 32 where we define events and input stream} A "definition file" or "input file" is a file, in which the statements and events for a test are stored. It directs generation of the test.
- [0030.]** [EM: delete this we have paragraph 33 instead] An "input stream" is a set of statements.
- 15   **[0031.]** A "directive" is a biasing component that can be attached to statements for test program generation in multiple areas.
- [0032.]** [EM: This should be after 33] An "event" is a combination of a condition and an input stream ~~or stream entity~~. The input stream is used to generate instructions if the condition is satisfied.
- 20   **[0033.]** An (this is an instruction statement) "instruction stream" or "input stream" is a sequence of partially specified instructions, which acts as a template for the test program to be generated.

### **Architectural Overview.**

- 25   **[0034.]** Reference is now made to Fig. 1, which is a block diagram of a test program generator that is operable in accordance with a preferred embodiment of the invention. A design verification system 10, used for verifying

a software or hardware design, has several basic interacting components. [EM: is this line important] Those components of the design verification system 10 that are located above a broken line 12 are dependent on the specification of the design being verified, while those located below the line 12 are independent of the specification.

**[0035.]** The design verification system 10 enables the creation of tests that have various degrees of randomness. The ability of the design verification system 10 to introduce random unspecified parameters is fundamental, since design flaws in practice are usually unpredictable.

10 **[0036.]** A [EM: Arthur I hope I'm not driving you crazy, but after speaking to others it may be better to talk about architectural models. Similarly, architectural (rather than behavioral) simulator] architectural specification model 14 holds a formal description of the specification of the system. This specification may be stored in a database, which may also incorporate testing knowledge of the system design. The integration of all the information stored in the specification model 14 is referred to herein as the knowledge base of the test program generator. The knowledge base typically includes heuristics, which represent test engineering expertise, and the knowledge base is designed to allow expert users to add knowledge.

20 **[0037.]** The content of an input file 20 is the input to a generic test program generator engine 22. The input file 20 is the main medium through which the test program generator engine 22 is influenced. This influence includes, for example, the identity of the test instructions, their relative order, and various events relating to the instructions. The test program generator engine 22 incorporates an event mechanism 23 for detecting and processing events in the input file 20.

**[0038.]** It is an important aspect of the invention that alternative input streams can be defined. These input streams are reactive to conditions that could occur during the execution of a test. In some cases, an input stream may exploit some condition, while in other cases the input stream may be designed to avoid the occurrence of a condition. Adaptive test program generation is achieved in the design verification system 10 by event detection and handling using the event mechanism 23.

**[0039.]** An event is represented by a condition and a set of statements that is generated only if the condition is satisfied. ~~some defined condition~~ holds. It is an important feature of the present invention that events are definable, and are recognized by the test program generator engine 22, and processed therein. An advantage of introducing events into the test program generation process is the capability of extensively adapting the behavior of the test program generator engine 22 to unexpected situations in a controlled manner, even though a high degree of randomness can be present in the generation of test instructions.

**[0040.]** In addition to receiving input via the input file 20, the test program generator engine 22 receives input information from the architectural specification model 14. The test program generator engine 22 generates test programs 24, during which process instructions are executed on a architectural behavioral simulator 26, and ultimately executed on a target or design 28, which can be a simulator. This produces results 29. The test program generator engine 22 may receive some generic knowledge of the specification, and can exploit this knowledge so as to generate sequences of instructions.

**[0041.]** The output of the test program generator engine 22 is a sequence of instructions. In a dynamic mode of operation, each instruction is first generated by the test program generator engine 22 and is then simulated by the architectural behavioral simulator 26. Generation typically involves selecting the instruction, the operands and the resources, and initializing the resources. As this is done, the instruction built by the test program generator engine 22 is passed to the architectural behavioral simulator 26 simulator, and eventually to the design 28 for execution. This cycle continues with the next instruction.

**[0042.]** The behavioral simulator 26 is used to predict the results of instruction execution in accordance with the specification of the system being verified. Any conventional behavioral simulator is suitable for the behavioral simulator 26.

**Input file Details.**

**[0043.]** Reference is now made to Fig. 2, which is a high level block diagram of an input file in accordance with a preferred embodiment of the invention. Input files are a preferred medium for practicing the present invention. The input file 20 is essentially a list of statements 30 for controlling the test output. There are various types of statements. The statements 30 may include, an event 34, and conventional instructions (not shown) that are unrelated to events.

**Instructions.**

**[0044.]** Instructions are the most basic statements in the input file 20. Instructions may be modified by operand biasing, biasing functions or directives.

**Statements.**

**[0045.]** The statements 30 in the input file 20 actually constitute a program that is interpreted by the test program generator engine 22. The programming language includes constructs that are used to control the operation of the test program generator engine 22. The language includes conditional statements, which can reference the current state of the program generated by the test program generator engine 22. These control constructs are not themselves generated.

**Events.**

**[0046.]** Reference is now made to Fig. 3, which is a block diagram illustrating the structure of an event 34 (Fig. 2) which is operative in accordance with a preferred embodiment of the invention. The event 34 has a group of specialized attributes: an identifying name 36, a triggering condition 38, a priority value 40 and an input stream or body 42 that defines instructions to be generated.

**[0047.]** The name 36 is simply the identifier of the event. It is a required attribute.

**[0048.]** The triggering condition 38 is a condition that is checked to decide whether the body 42 should be used to generate instructions. It is a required attribute. In some cases the triggering condition 38 can be given the value TRUE, in which case the body 42 is always used.

**[0049.]** In some embodiments, events may be specified directly in the input files. In other embodiments, events may be stored in separate event files, which are included by the input file 20. It is also possible for event files to include other event files.

**[0050.]** Reference is now made to Fig. 4, which is a block diagram illustrating the test generator engine in further detail, in accordance with a preferred embodiment of the invention. The test program generator engine 22 (Fig. 1) maintains a list of events 44. Each of the events 44 includes a name 46, and is used for purposes of identification, a triggering condition 48, a body 50, which is an alternative instruction stream to be generated when the event occurs, and further includes a relative priority value 52. Evaluation of triggering conditions by the test program generator engine 22 occurs in order of the priority value 52.

5  
10 **[0051.]** Listing 1 is a high level procedure which is encoded into the test program generator engine 22 for the detection and handling of events that are defined in the input file.

Listing 1

Procedure: HandleEvents

15 Input: currentStatement

1. For each event eh in order of priority

// check if event is ~~disabled~~suppressed

2. If !currentStatement -> ~~Disables~~Suppresses(eh)

20

//evaluate triggering condition

3.if eh->ConditionIsTriggered()

25

//generate the instruction stream with the

//event as specified by its body

4. eh->GenerateInstructionStream()

**[0052.]** The procedure HandleEvents in Listing 1 is called by the test program generator engine 22 before each instruction is generated.

**[0053.]** Design flaws are often exposed only when two or more events occur concomitantly. For example, a program interrupt occurring while another interrupt is being handled, or following a false branch prediction. In order to expose such flaws, the procedure `HandleEvents` is designed to handle events recursively. When nesting of events occurs, the processing of the primary event is paused while the secondary event is handled.

**Examples.**

**[0054.]** The following two examples are useful in explaining the operation of an event according to the invention. In Listing 2 and Listing 3, the body of the named event is a single instruction. However the body of an event is an input stream, and not limited to a single instruction.

**Branch-on-zero event.**

**[0055.]** A branch-on-zero event occurs whenever the zero bit of a particular conditional register is set. Recognition of the branch-on-zero event by the test program generator engine 22 (Fig. 1), using one of the events 44 (Fig. 4), results in the generation of a conditional branch. The particulars of the event for the branch-on-zero event are shown in Listing 2.

Listing 2

```
Name: BranchOnZero
20 //Check if the zero bit of the condition register
   // is set
   cond: ConditionRegisterZero = 1

   //issue a conditional branch on zero instruction
25 body:
   bez
```

**Escape Sequence Event.**

**[0056.]** An escape sequence event occurs whenever there is insufficient uninitialized memory at the address currently held in the program counter (PC). Recognition of the escape sequence event by the test program generator engine 22, using another one of the events 44, causes the generation of an unconditional branch to an unoccupied memory location using the event mechanism 23. The particulars of the event for the escape sequence event are shown in Listing 3.

10 Listing 3  
name: EscapeSequence  
  
//check that there is sufficient room to write  
//a branch instruction, but nothing else  
15 cond: !IsInitialized (\$PC) AND IsInitialized (\$PC+4)  
  
//generated branch instruction  
body:  
    br  
20

**Interrupt Handler.**

**[0057.]** The event shown in Listing 4 is typical of an interrupt handler generated by the test program generator engine 22 (Fig. 1). It responds to an event entitled "DSI", which involves an interrupt, and includes instructions that return the program counter to the address immediately following the instruction that caused the interrupt. The triggering condition for the event is that the program counter contains the address (or addresses) appropriate to the interrupt.

15

## Listing 4

Name: DSI\_Interrupt\_Handler

```
5      //check that DSI exception occurred
      cond: $PC == 0x0000000000000300 ||
           $PC == 0x0000000000000304

      //generated handler instructions
10     body:
           mfsrr0 G28
           addi G28,G28,0x4
           mtsrr0 G28
           mtdar G28
15     rfid
```

**Operation.**

**[0058.]** Reference is now made to Fig. 5, which is a flow chart illustrating a method of test program generation, which is operative in accordance with a preferred embodiment of the invention. The description of Fig. 5 should be read in conjunction with Figs. 1, 2, 3, and 4. The test program generation cycle begins at initial step 54, in which the design verification system 10 (Fig. 1) is configured for testing the system design. Appropriate files are loaded in order to initialize the test program generator engine 22, the specification model 14, the behavioral simulator 26, and the design 28.

**[0059.]** Next, at step 60, the test program generator engine 22 reads the input file 20. In particular, each event 34 is interpreted by the test program generator engine 22, and stored in a list of events 44 (Fig. 4). [EM: I'm assuming that this implies that the main input stream is also read.] For purposes of this disclosure, only the processing of events is disclosed with reference to Fig. 5, it being understood that many statements unrelated to events

may also be read from the input file 20, which would be processed conventionally.

**[0060.]** The list of the events 44 that were developed in step 60 are scanned in order of priority. At step 70 the member of the list of events 44  
5 having the highest priority, and which is as yet unevaluated is selected.

**[0061.]** Decision step 72 is an optional step that is performed in alternate embodiments where events may be disabled. Decision step 72 can be omitted, in which case control passes directly from step 70 to decision step 76.

**[0062.]** At decision step 72 a test is made to determine whether the event  
10 selected in step 70 is currently disabled. If the determination at decision step 72 is affirmative, then control proceeds to decision step 76.

**[0063.]** If the determination at step 70 is negative, then control proceeds to decision step 74.

**[0064.]** At decision step 74 the event selected in step 70 is evaluated to  
15 determine if its triggering condition 48 currently holds. The triggering condition 48 is a Boolean expression that is evaluated by the test program generator engine 22, which may refer to current resource values. In such a case the test program generator engine 22 must refer to the behavioral simulator 26 in order to access these values. For example, if the test program generator engine 22 evaluates the condition of the event given in Listing 4, it  
20 needs to know the value of the program counter register (\$PC), and must obtain this value from the behavioral simulator 26.

**[0065.]** If the determination at decision step 74 is negative, then control proceeds to decision step 76, which is disclosed below.

**[0066.]** If the determination at decision step 74 is affirmative, then control  
25 proceeds to step 78. An alternative instruction stream, specified in the

body 50 of the current one of the events 44, is used as a template to generate instructions in the same manner as the primary instruction stream. Step 78 is typically implemented recursively. The processing of the primary instruction stream is suspended until **the ALTERNATE INSTRUCTION**

5 **STREAM BEGIN IN** step 78 is completed.

[0067.] [EM: This is wrong, after step 78, we continue with 62, but use the body of the event rather than the main instruction stream as the input stream. This is important.] Upon completion of step 78, generation of the primary instruction stream is resumed, correct this and control is transferred  
10 to step 62, which is disclosed below.

[0068.] At decision step 76, it is determined if more events remain to be evaluated. If the determination at decision step 76 is affirmative, then control returns to decision step 70 to process the next event.

[0069.] If the determination at decision step 76 is negative, then control  
15 proceeds to step 62.

[0070.] At step 62 the generation of an instruction stream is begun. A current state of the design is determined, using the behavioral simulator 26. Control now proceeds to step 64.

[0070a.] There should be a step between 62 and 64 which gets the next instruction statement from the input stream.  
20

[0071.] At step 64, the test program generator engine 22 proposes an instruction. The proposed instruction may be random, depending upon the current test program generation policy in force, and subject to any rule constraints that may be in effect.  
25

**[0072.]** Next, at step 66, the proposed instruction is evaluated in the behavioral simulator 26. The states of the design existing immediately prior to execution of the instruction, and the predicted state following execution of the instruction are then both known.

- 5 **[0073.]** Next, in step 68, the test program generator engine 22 outputs the instruction as part of the test program generated that was proposed at step 64 into the instruction stream. Control passes to decision step 80.

- [0074.]** At decision step 80, it is determined if more instructions need to be generated in the test. If the determination at decision step 80 is affirmative, then control returns to step 70.

10 **[0075.]** If the determination at decision step 80 is negative then control proceeds to final step 82, and the current phase of the test program generation cycle is complete.

**Alternative Modes of Operation.**

- 15 **[0076.]** In some embodiments, for any instruction, or sequence of instructions in the input stream, it may be specified that detection and activation of a particular event should be disabled during test program generation. This provides enhanced control of the test program generation process.

- 20 **[0077.]** While this invention has been explained with reference to the structure disclosed herein, it is not confined to the details set forth, and this application is intended to cover any modifications and changes as may come within the scope of the following claims:

What is claimed is.

1. A system for verification of a system design, comprising:
  - 5 a test program generator that accepts a sequence of statements including at least one event;  
an event handling facility in said test program generator; and  
wherein responsive to a triggering condition of said event said test program generator emits TEST INSTRUCTIONS RESPONSIVE to one of a primary instruction stream and an alternate instruction stream. said alternate instruction stream being represented in a body of said event.
  - 10
2. The system according to claim 1, wherein said event comprises a plurality of events that are processed in order of priority values thereof in said event handling facility.
  - 15
3. The system according to claim 1, wherein said body of said event comprises program instruction statements that are interpreted by said test program generator.
  - 20
4. The system according to claim 3, wherein a conditional statement of said event program instructions references a current state of a program that is generated said test program generator.
- 25 5. A method of test program generation, comprising the steps of:  
defining a set of statements, said set of statements including an event;

responsive to said set of statements generating a test program of instructions stream of instructions for a target;

while performing said step of generating said stream of instructions determining if a condition of said event is satisfied; and

- 5 responsive to said step of determining generating an alternate test program of instructions stream of instructions.

6. The method according to claim 5, wherein said step of determining is performed by evaluating a state of said target prior to inclusion of an instruction  
10 in said stream of instructions.

7. The method according to claim 6, wherein said step of evaluating said state is performed prior to a simulated execution of said instruction.

- 15 8. The method according to claim 6, wherein said step of evaluating said state is performed subsequent to a simulated execution of said instruction.

9. The method according to claim 6, wherein said step of evaluating said state is performed a first time prior to a simulated execution of said instruction and is performed a second time subsequent to said simulated execution  
20 thereof.

10. The method according to claim 5, wherein at least a portion of said stream of instructions are randomly generated selected.  
25

11. The method according to claim 5, wherein said event comprises a plurality of events, each of said events having a priority value, and said step of determining if said condition is satisfied is performed with respect to each of said events in an order of said priority value thereof.
- 5
12. The method according to claim 5, wherein said event has an identifying name attribute.
13. The method according to claim 5, wherein said event has a triggering
- 10 condition attribute.
14. The method according to claim 5, wherein said event comprises a input stream stream entity.
- 15
15. The method according to claim 5, wherein said event comprises an identifying name attribute, a triggering condition attribute, and a stream entity. and a priority value.
16. A method of test program generation, comprising the steps of:
- 20 providing a test program generation engine;
- coupling said test program generation engine to a design specification of a target, wherein said design specification comprises a knowledge base;
- coupling said test program generation engine to a behavior simulator of said target;
- 25 introducing a set of statements into said test program generation engine, said set of statements including an event;

## 5

determining whether a triggering condition of said event is satisfied;  
in a first case, wherein said triggering condition is satisfied, causing said test  
program generation engine to generate a test instructions based on first  
stream of instructions that can be executed on said target; and  
5 in a second case, wherein said triggering condition is not satisfied, causing  
said test program generation engine to generate a second stream of instruc-  
tions that can be executed on said target.

10 17. The method according to claim 16, wherein at least a portion of said in-  
struction is generated randomly.

15 18. The method according to claim 16, wherein said step of determining is  
performed by evaluating a state of said target prior to inclusion of said in-  
struction in said first stream of instructions.

19. The method according to claim 18, wherein said step of evaluating said  
state is performed prior to said simulated execution of said instruction.

20 20. The method according to claim 16, wherein said event comprises a plu-  
rality of events, each of said events having a priority value, and said step of  
determining is performed with respect to each of said events in an order of  
said priority value thereof.

25 21. The method according to claim 16, wherein said set of statements is in-  
troduced into said test program generation engine as an input file.

6

22. The method according to claim 16, wherein said event has an identifying name attribute.

5 23. The method according to claim 16, wherein said event has a triggering condition attribute.

24. The method according to claim 16, wherein said event comprises a stream entity.

10 25. The method according to claim 16, wherein said event comprises an identifying name attribute, a triggering condition attribute, and a stream entity.

15 26. A computer software product, comprising a computer-readable medium in which computer program instructions are stored, which instructions, when read by a computer, cause the computer to generate test programs by performing the steps of:

20 accepting a set of statements, said set of statements including an event; responsive to said set of statements generating a stream of instructions for a target; while performing said step of generating said stream of instructions determining if a condition of said event is satisfied; and responsive to said step of determining generating an alternate stream of instructions.

25

27. The computer software product according to claim 26, further comprising the steps of accessing a knowledge base having information of said target stored therein, and said step of generating said stream of instructions comprises selecting members of said stream of instructions in accordance with  
5 said information in said knowledge base, wherein said step of selecting is biased by said set of statements.

28. The computer software product according to claim 26, wherein at least a  
10 portion of said stream of instructions are randomly selected.

29. The computer software product according to claim 26, wherein said event comprises a plurality of events, each of said events having a priority value, and said step of determining if said condition is satisfied is performed with respect to each of said events in an order of said priority value thereof.  
15

30. The computer software product according to claim 26, wherein said event has an identifying name attribute.

31. The computer software product according to claim 26, wherein said event  
20 has a triggering condition attribute.

32. The computer software product according to claim 26, wherein said event comprises a body that is a template for generation of said alternate stream of instructions.  
25

33. The computer software product according to claim 26, wherein said event comprises an identifying name attribute, a triggering condition attribute, and a stream entity.

- 5 34. A computer software product, comprising a computer-readable medium in which computer program instructions are stored, which instructions, when read by a computer, cause the computer to generate test programs by performing the steps of:
- defining a test program generation engine in a memory;
- 10 defining a design specification of a target in said memory, wherein said design specification comprises a knowledge base;
- defining a behavior simulator of said target in said memory;
- coupling said test program generation engine to said design specification;
- coupling said test program generation engine to a behavioral simulator;
- 15 accepting a set of statements into said test program generation engine, said set of statements including an event;
- responsive to said set of statements, and to information in said knowledge base, causing said test program generation engine to generate an instruction that can be executed on said target;
- 20 thereafter determining in said behavior simulator whether a triggering condition of said event is satisfied by a simulated execution of said instruction;
- in a first case, wherein said triggering condition is satisfied, causing said test program generation engine to generate a first stream of instructions that can be executed on said target; and

in a second case, wherein said triggering condition is not satisfied, causing said test program generation engine to generate a second stream of instructions that can be executed on said target.

5     35. The computer software product according to claim 34, wherein at least a portion of said instruction is generated randomly.

36. The computer software product according to claim 34, wherein said step of determining is performed by evaluating a state of said target prior to inclusion of said instruction in one of said first stream of instructions and said second stream of instructions.

10

37. The computer software product according to claim 36, wherein said step of evaluating said state is performed prior to said simulated execution of said instruction.

15

38. The computer software product according to claim 36, wherein said step of evaluating said state is performed subsequent to said simulated execution of said instruction.

20

39. The computer software product according to claim 36, wherein said step of evaluating said state is performed a first time prior to a simulated execution of said instruction and is performed a second time subsequent to said simulated execution thereof.

25

10

40. The computer software product according to claim 34, wherein said event comprises a plurality of events, each of said events having a priority value, and said step of determining is performed with respect to each of said events in an order of said priority value thereof.

5

41. The computer software product according to claim 34, wherein said set of statements is introduced into said test program generation engine as an input file.

10

42. The computer software product according to claim 34, wherein said event has an identifying name attribute.

43. The computer software product according to claim 34, wherein said event has a triggering condition attribute.

15

44. The computer software product according to claim 34, wherein said event comprises a stream entity.

20

45. The computer software product according to claim 34, wherein said event comprises an identifying name attribute, a triggering condition attribute, and an input stream entity.

46. A program test generator, comprising:  
a test program generation engine;

a design specification of a target, wherein said design specification comprises a knowledge base, wherein said test program generation engine is coupled to said design specification;

5 a behavior simulator of said target coupled to said test program generation engine;

wherein said test program generation engine is adapted to accept a set of statements, said set of statements including an event;

responsive to said set of statements, and to information in said knowledge base, said test program generation engine generates an instruction that can  
10 be executed on said target;

wherein responsive to said instruction said test program generator (not the behavior simulator) determines whether a triggering condition of said event is satisfied by a simulated execution of said instruction;

responsive to said behavior simulator, in a first case, wherein said triggering  
15 condition is satisfied, said test program generation engine generates test instructions based on a first stream of instructions that can be executed on said target; and

in a second case, wherein said triggering condition is not satisfied, said test program generation engine generates a second stream of instructions that  
20 can be executed on said target.

47. The program test generator according to claim 46, further comprising a design simulator for simulating said simulated execution.

25 48. The program test generator according to claim 46, wherein at least a portion of said instruction is generated randomly.

49. The program test generator according to claim 46, wherein said behavior simulator determines whether said triggering condition of said event is satisfied by evaluating a state of said target prior to inclusion of said instruction in one of said first stream of instructions and said second stream of instructions.

50. The program test generator according to claim 49, wherein said step of evaluating said state is performed prior to said simulated execution of said instruction.

10

51. The program test generator according to claim 49, wherein said step of evaluating said state is performed subsequent to said simulated execution of said instruction.

52. The program test generator according to claim 49, wherein said step of evaluating said state is performed a first time prior to said simulated execution of said instruction and is performed a second time subsequent to said simulated execution.

53. The program test generator according to claim 46, wherein said event comprises a plurality of events, each of said events having a priority value, and said step of determining is performed with respect to each of said events in an order of said priority value thereof.

## 13

54. The program test generator according to claim 46, wherein said set of statements is introduced into said test program generation engine as an input file.

5 55. The program test generator according to claim 46, wherein said event has an identifying name attribute.

56. The program test generator according to claim 46, wherein said event has a triggering condition attribute.

10

57. The program test generator according to claim 46, wherein said event comprises a stream entity.

15 58. The program test generator according to claim 46, wherein said event comprises an identifying name attribute, a triggering condition attribute, and a stream entity.

## 2

## Adaptive Test program generation

**ABSTRACT OF THE DISCLOSURE**

5

A test program generator that produces test instructions according to a specification of a system being verified. The instructions are typically generated randomly, at least in part, and are then. The system is capable of interpreting events, detecting an impending occurrence of an event, and responding to the event by generating an alternate instruction stream.

10